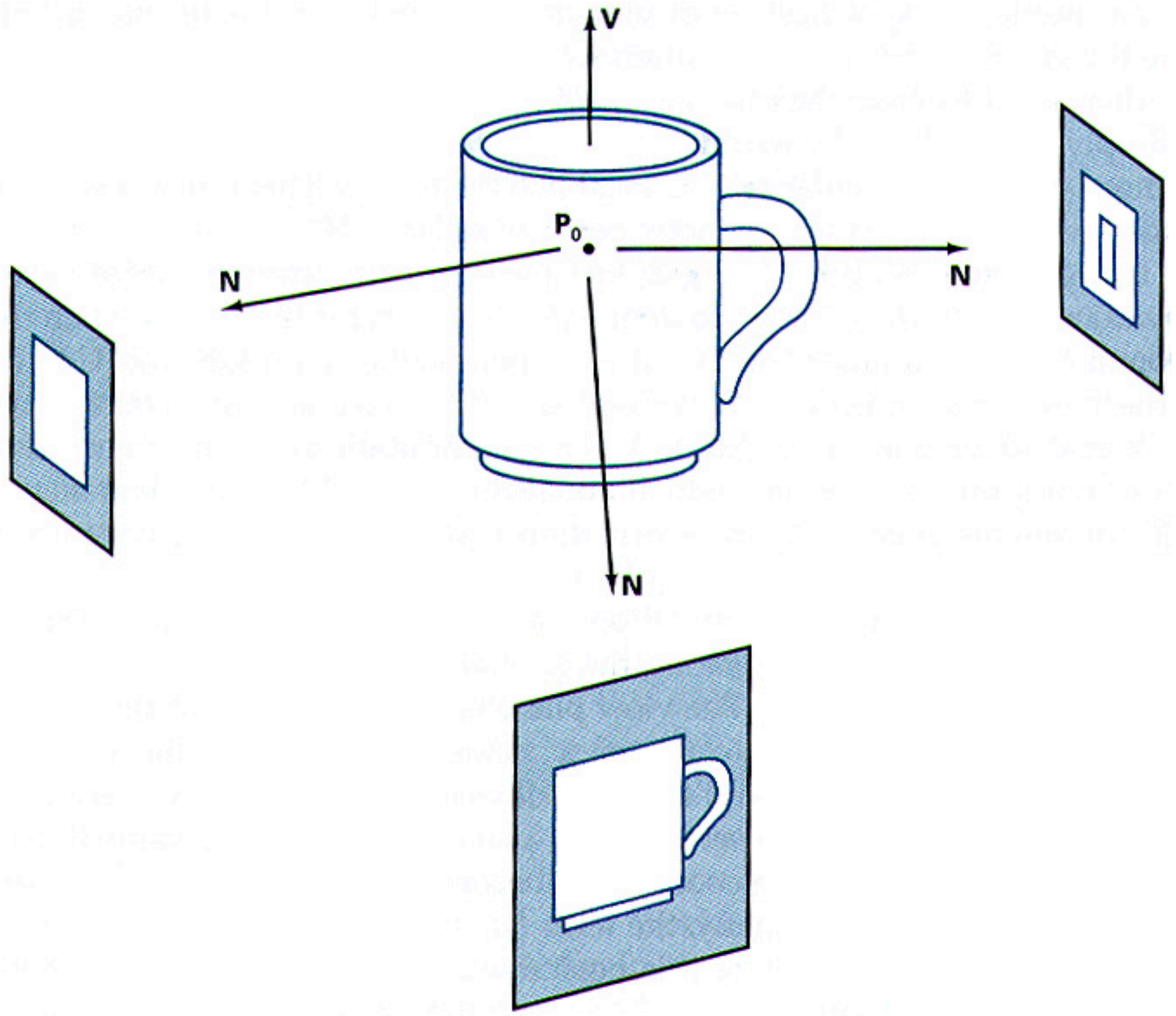


Viewing a scene from different directions with a fixed view-reference point.

Monash University • Clayton's School of Information Technology

CSE3313 Computer Graphics

Lecture 16: Compound Modelling Transformations and Projections

Modelling Transformations

- Suppose our application program draws cars.
- A low level routine might be called to draw a wheel.
- There is no special place for a wheel to appear in the scene, so the wheel drawing routine draws the wheel in a **modelling** coordinate system.
- The modelling routine that draws the car calls the routine to draw the wheel at least four times. Each time a wheel is drawn it can be scaled, translated and rotated to fit in a particular part of the full model, thus it can have a single transformation matrix of its own to indicate how the modelling coordinates can be turned into world coordinates.
- The routine to draw a car draws the car in its own modelling coordinate system.

Modelling Transformations (cont.)

- A routine which draws a road may call the routine to draw a car multiple times. Each *instance* where a car gets drawn gets its own transformation matrix.

- A wheel object is transformed to place it in the car object and all the car objects are transformed when the car object is placed in the scene. The overall effect is as follows:

$$[\text{wheel1}]_{\text{world coords.}} = M_{\text{wheel1}}[\text{wheel}]_{\text{modeling coords.}}$$

$$[\text{wheel2}]_{\text{world coords.}} = M_{\text{wheel2}}[\text{wheel}]_{\text{modeling coords.}}$$

.

.

.

- At any stage, modelling transformation that gets carried out on the initial modelling coordinates is a product of separate modelling transformation matrices.

The Current Transformation Matrix

- Many APIs have the concept of a **current transformation matrix (CTM)**. This transformation matrix is applied to any vertex.
- By default the CTM is the identity matrix.
- Functions for changing the CTM are often of two forms:
  - (1) functions that reset the CTM to some matrix;
  - (2) functions that pre-multiply or post-multiply the CTM by some new transformation matrix.
- OpenGL only uses post-multiplication.
- Suppose the $\mathbf{CTM}$ is post-multiplied by $\mathbf{T}_{\mathbf{new}}$

$$\mathbf{CTM}_{\mathbf{new}} \leftarrow \mathbf{CTM}_{\mathbf{old}}\,\mathbf{T}_{\mathbf{new}}$$

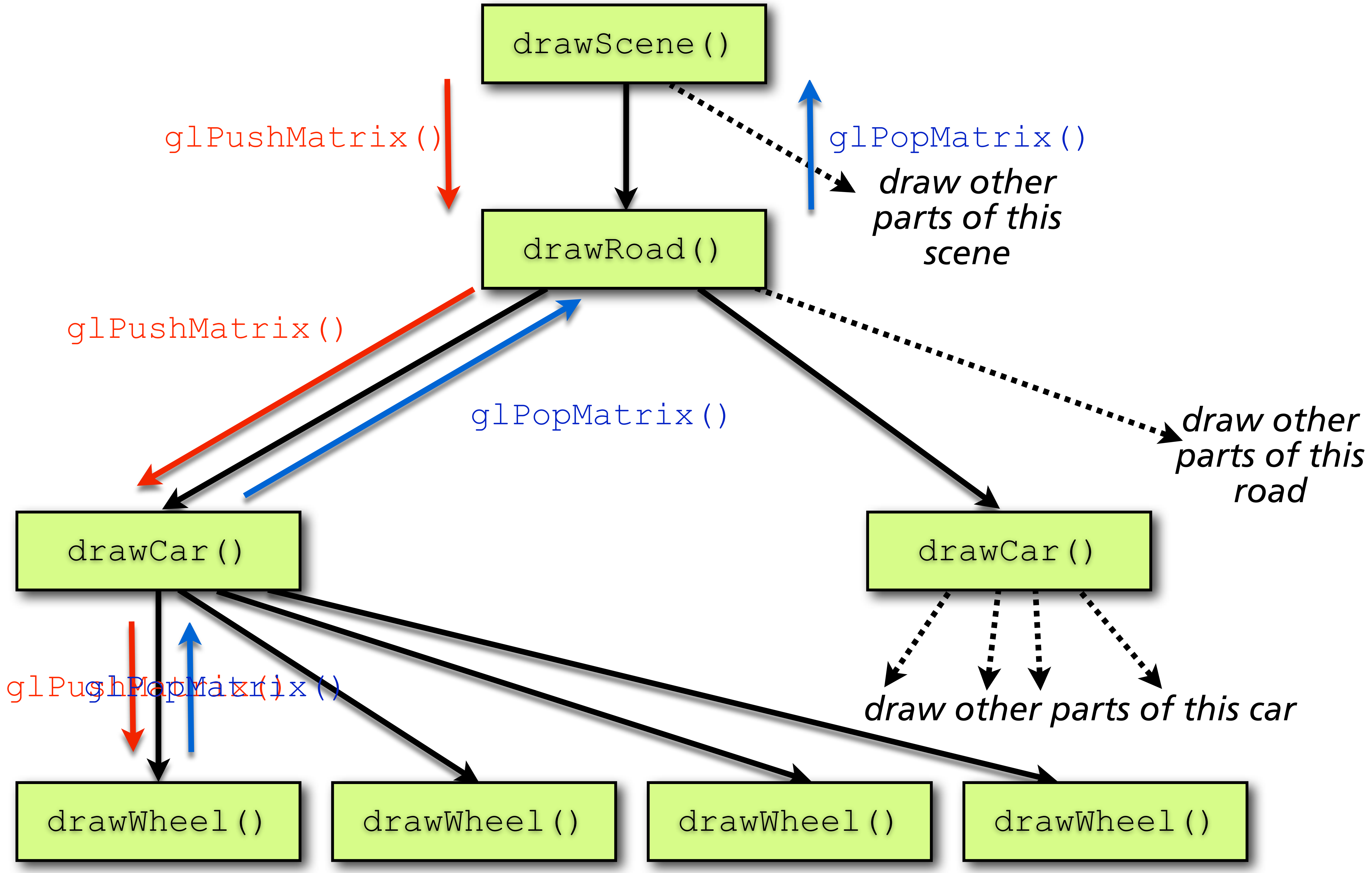
- The effect of $\mathbf{CTM}_{\mathbf{new}}$ is to carry out $\mathbf{T}_{\mathbf{new}}$ first and then to carry out $\mathbf{CTM}_{\mathbf{old}}$.

The Current Transformation Matrix (cont.)

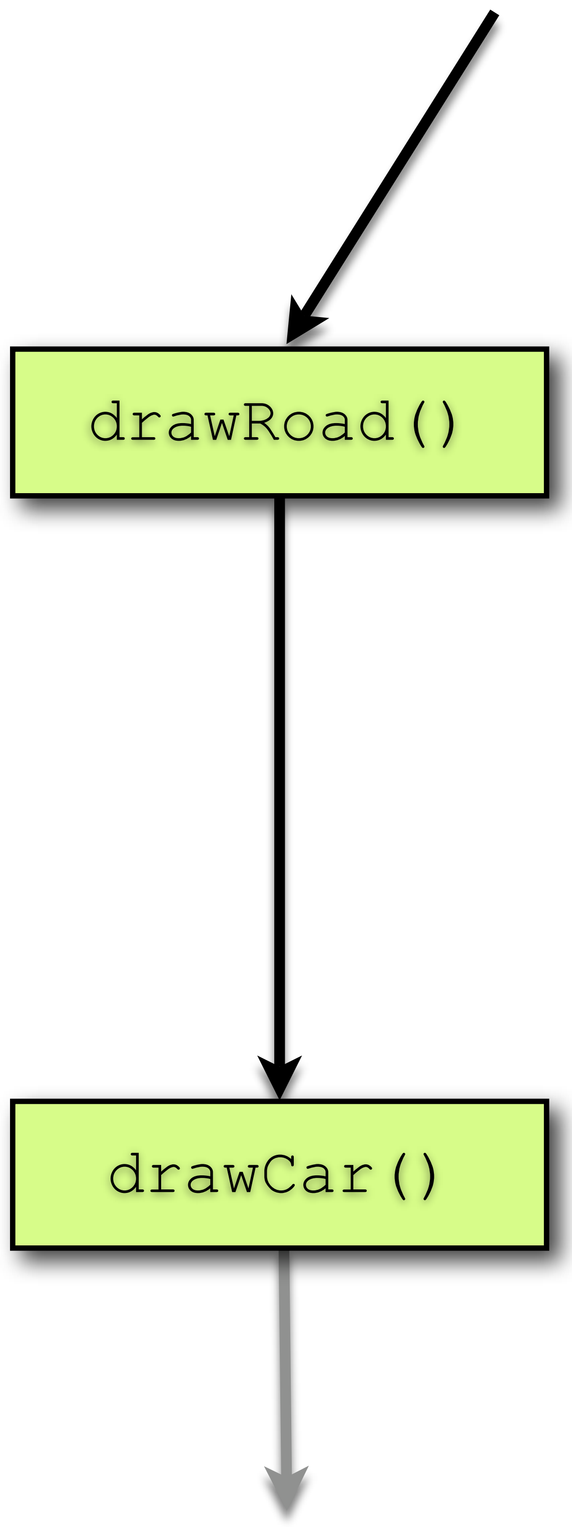
- Consider what happens with hierarchical modelling. Suppose we call a routine to draw a car which positions the car using $\mathbf{T}_{\mathbf{car}}$.
- This routine post-multiplies the CTM by $\mathbf{T}_{\mathbf{car}}$.
- The routine to draw the car calls the routine to draw a wheel.
- The wheel drawing routine positions the wheel, within the car, using $\mathbf{T}_{\mathbf{wheel1}}$ so that the CTM becomes:

  $$\mathbf{CTM}\ \mathbf{T}_{\mathbf{car}}\ \mathbf{T}_{\mathbf{wheel1}}$$

- When this new CTM is applied to the vertices of *wheel*, $\mathbf{[wheel]}_{\mathbf{wheel\ coordinates}}$, they are transformed to the appropriate place in the car's modeling coordinates via $\mathbf{T}_{\mathbf{wheel}}$ and the resulting coordinates, $\mathbf{T}_{\mathbf{wheel1}}\mathbf{[wheel]}_{\mathbf{wheel\ coordinates}}$ are transformed by $\mathbf{T}_{\mathbf{car}}$ to appear where the car should appear in the scene.


drawScene()
glPushMatrix()
glPopMatrix()
draw other parts of this scene
drawRoad()
glPushMatrix()
glPopMatrix()
draw other parts of this road
drawCar()
drawCar()
glPushMatrix()
glPopMatrix()
draw other parts of this car
drawWheel()
drawWheel()
drawWheel()
drawWheel()

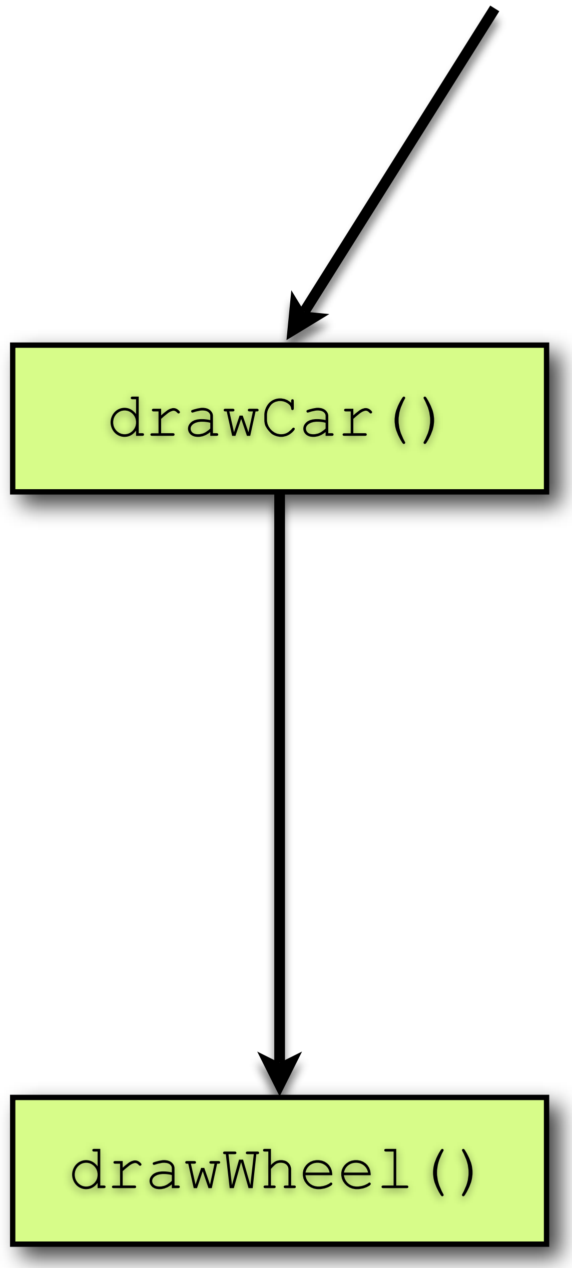
Car Drawing Example



```
drawRoad()
{
  draw road geometry
  for i = 1 to NUM_CARS
  {
    glPushMatrix();
    load car i transform
    drawCar(i);
    glPopMatrix();
  }
  return;
}
```

```
drawCar()
{
  ...
```

Car Drawing Example



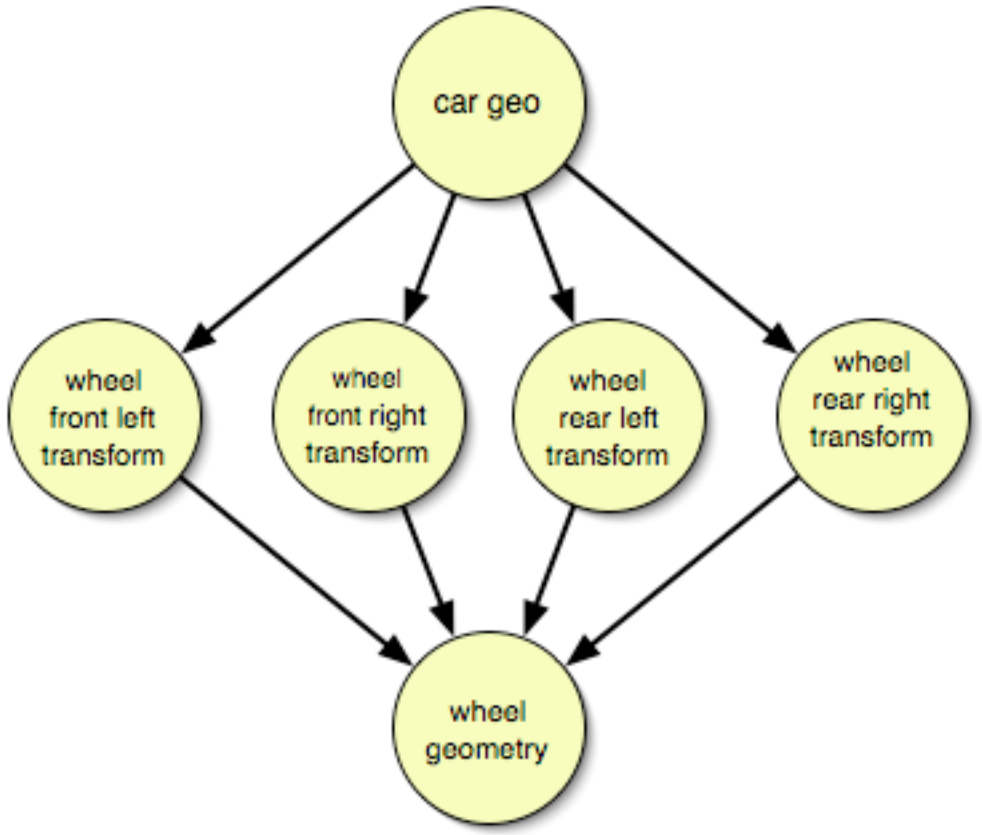
```
drawCar()
{
  draw car geometry
  for i = 1 to 4
  {
    glPushMatrix();
    load wheel *i* transform
    drawWheel(*i*);
    glPopMatrix();
  }
  return;
}
```

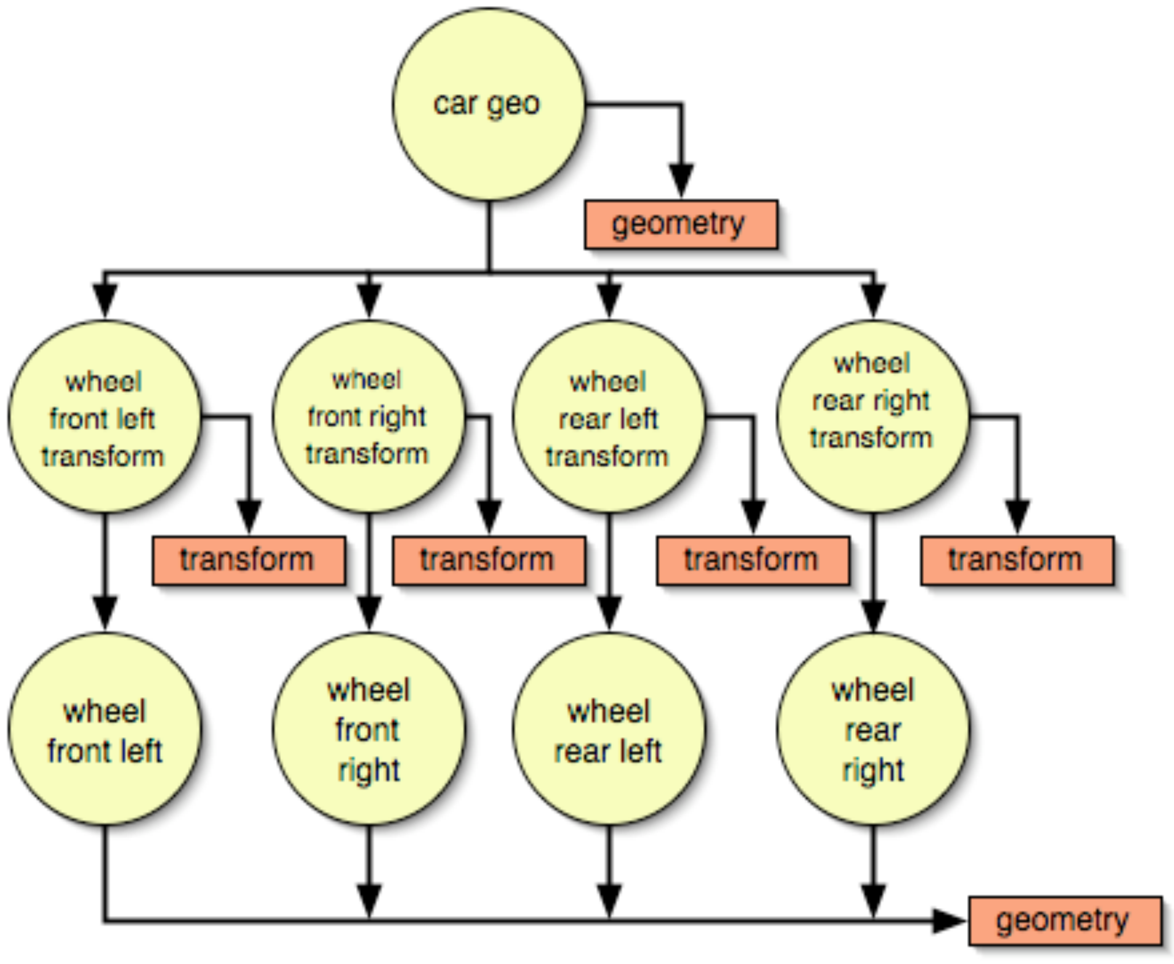
```
drawWheel()
{
  draw wheel in local co-ordinates
  return;
}
```

Scene Graphs

- A *scene graph* is a generalised structure for drawing 3D scenes
- Nodes can represent geometry, , lights, cameras, transformations, grouping, state changes (e.g. rendering mode), even animation drivers.
- Nodes are connected together to form a graph representing the scene.
- Nodes in the graph are traversed from the *root node* in order to render the scene.
- A much higher level of abstraction and usually more efficient than direct OpenGL calls. Functionality such as bounding box culling can reduce geometry sent to the graphics card.

Scene Graphs (cont.)


car geo
wheel front left transform
wheel front right transform
wheel rear left transform
wheel rear right transform
wheel geometry



Conceptual Scene Graph vs. implementation for OpenSG

Some Scene Graphs:

OpenSG - www.opensg.org
OpenSceneGraph - http://www.openscenegraph.org
Coin - www.coin3d.org (based on OpenInventor)
Simple Scene Graph - plib.sourceforge.net/ssg/index.html

Matrix Manipulation

- In OpenGL the matrix that is applied to all primitives is the product of the model-view matrix and the projection matrix. We can think of the CTM as the product of these two matrices. We can change each matrix individually by setting the mode using `glMatrixmode`.

- A matrix can be loaded via the function

  ```
      glLoadMatrix( pointer_to_matrix );
  ```

  or we can set it to the identity via

  ```
      glLoadIdentity( );
  ```

- The matrix we load is a one dimensional array of 16 entries, stored by columns. We can use this to post-multiply the selected matrix by

  ```
      glMultMatrix( pointer_to_matrix );
  ```

- We can get the contents of the current matrix using

  ```
  glGetFloatv(GL_MODELVIEW_MATRIX, pointer_to_matrix);
  ```

Rotation, Translation and Scale

- Rotation, translation and scaling are provided via:

```
glRoatatef( angle, vx, vy, vz);
glTranslatef( dx, dy, dz );
glScalef( sx, sy, sz );
```

- For rotation, `vx`, `vy`, and `vz` are the components of the vector about which we wish to rotate.
- In the translation function, `dx`, `dy` and `dz` are the components of the displacement vector.
- For scaling, `sx`, `sy` and `sz` are the scaling factors along the x, y and z coordinate axes respectively.
- Transformations must be specified in the *reverse order* to that which we want them performed.

Pushing and Popping Matrices

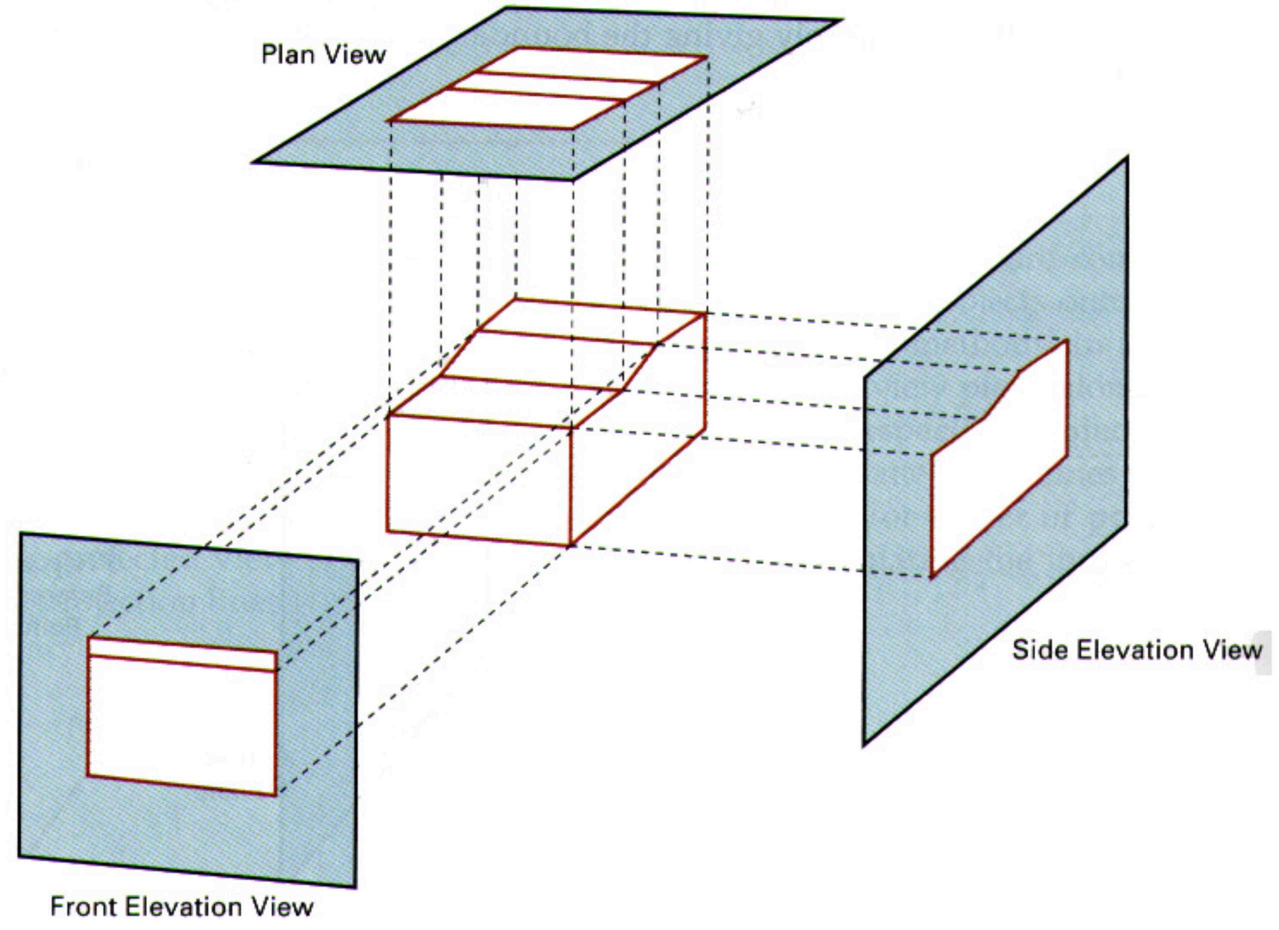
- Sometimes we want to perform a transformation and then return to the same state as before the transformation.
- Rather than undo the previous transformation by post-multiplying by its inverse or recomputing the old CTM we can push the CTM onto a stack. e.g.

```
glPushMatrix( );
   glTranslatef( ... );
   glRotatef( ... );
   glScalef( ... );
   /* draw the object ... */
glPopMatrix();
```

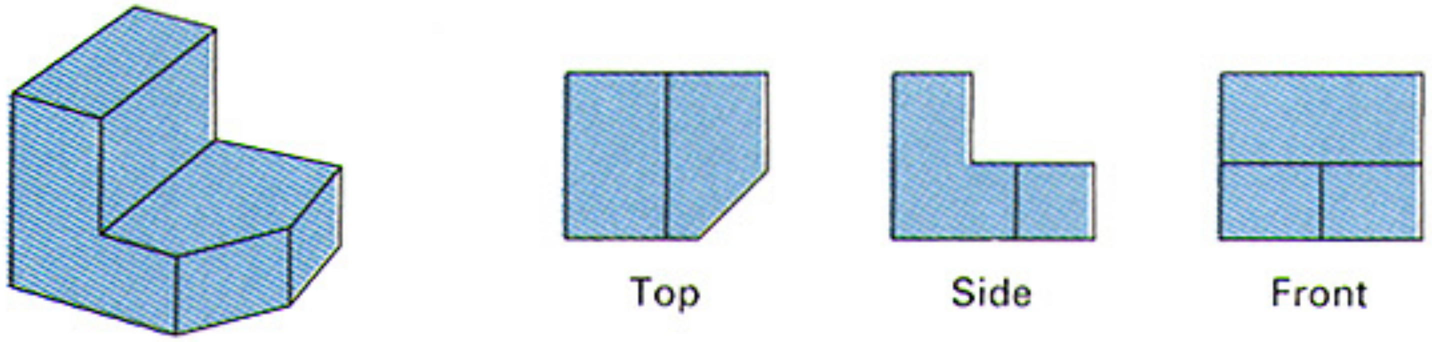
Parallel Projections

- Views formed by parallel projections are characterized by the angle the direction of projection makes with the projection plane.
- If the angle of projection is perpendicular to the projection plane, we have an **orthographic projection**, otherwise we have an **oblique projection**.
- Orthographic projections are often used to provide front, side and top views of an object. Orthographic front, side and rear views are often called *elevations* while top views are called *plans.*
- Engineering drawings (CAD/CAM) commonly employ orthographic projections since lengths and angles are accurately depicted and can be measured from the drawings.
- Orthographic views which show more than one face of an object are called **axonometric** views.

Orthographic Projections

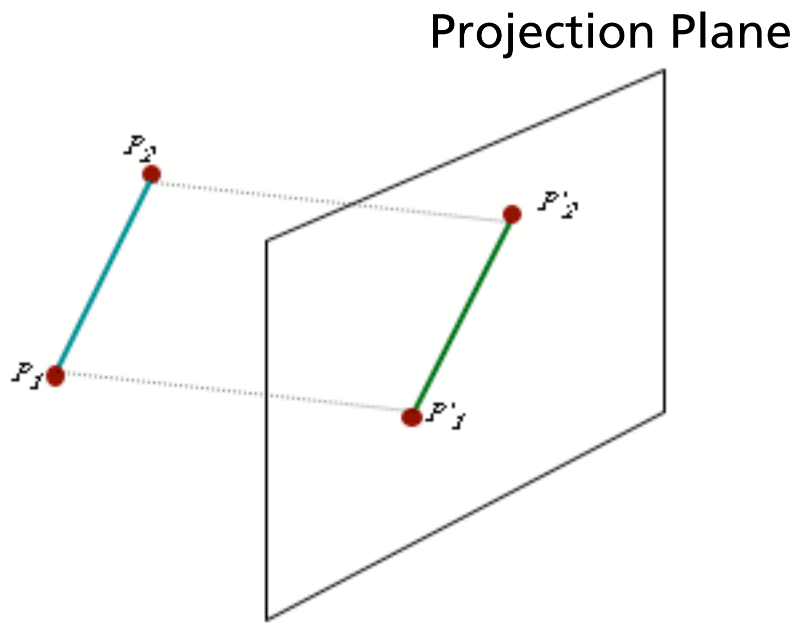


Orthographic projections of an object, displaying plan and elevation views.



Three parallel-projection views of an object, showing relative proportions from different viewing positions.

Orthographic Transformation



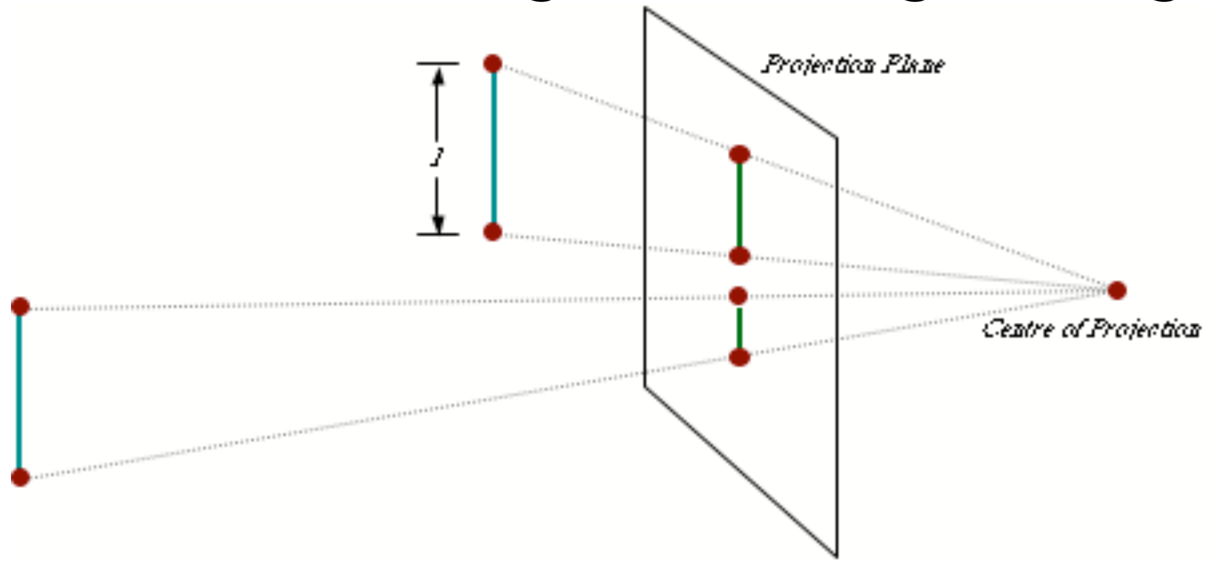
$$M_{ortho} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- Transformation equations are straightforward for orthographic parallel projections.
- If the projection plane is $z = 0$, for any point (x,y,z) the projection point on the viewing surface is (x_p, y_p, z_p) with

  $x_p = x,\ y_p = y,\ z_p = 0.$
- The transformation matrix is $\mathbf{M}_{\mathbf{ortho}}$. This transformation cannot be reversed: information about the z coordinate is lost.

Perspective Projection

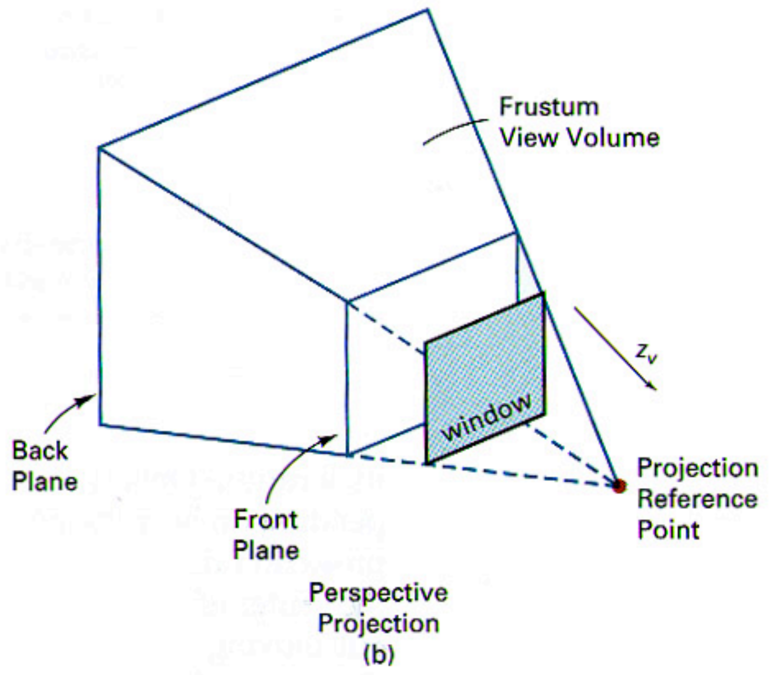
- All perspective views are characterised by a diminution of size.
- As objects move further from the viewer, their images become smaller. This is an example of a **depth cue**. I.e. We have a 2D picture but we can deduce missing depth information from information contained in the 2D image.
- This size change means we cannot use perspective views to get measurements of line lengths as in engineering drawings.

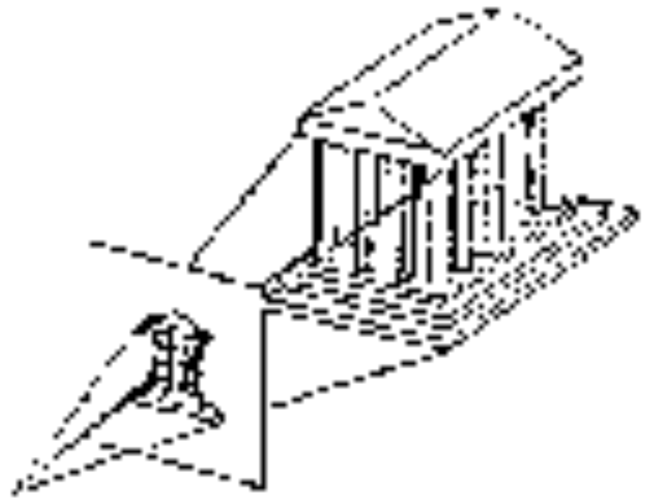


Perspective projection of lines of equal length

Perspective Projection (cont.)

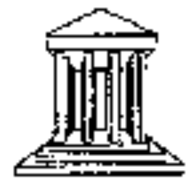
- In classical perspective views, the viewer is located at the apex of a symmetrical pyramid which is determined by a window in the projection plane.
- The pyramid shape determines the view volume for clipping. It is often termed the frustum of vision or viewing frustum.
- The view volume for clipping is different for parallel and

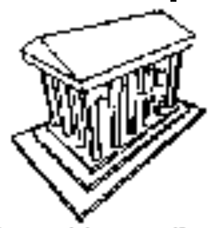


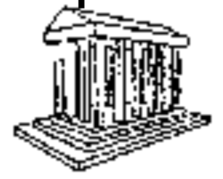


Perspective Projection (cont.)

- The classical perspective views as used in art are usually known as one–, two– and three–point perspectives.
- Parallel groups of lines that are parallel to the projection plane appear parallel in projection, while groups of parallel lines that are not parallel to the projection plane will converge at a vanishing point.
- 1–, 2– and 3–point perspectives have 1, 2 or 3 vanishing points respectively. Technically the number of vanishing points is determined by the number of principle axes of the world coordinate system that intersect with the projection plane.
- With hand drawn perspectives, higher point perspectives are more difficult to achieve, however in computer graphics all perspectives can be achieved using the general perspective projection.


One-point perspective


Three-point perspective


Two-point perspective